MINAR International Journal of Applied Sciences and Technology

Article type : **Research Article**

Date Received : **22/07/2020**
Date Accepted : **03/08/2020**
Date published : **01/09/2020**
: **www.minarjournal.com**

DESIGN AND IMPLEMENTATION OF KRUSKAL'S MINIMUM SPANNING TREE ALGORITHM IN C++

Nadia Moqbel Hassan ALZUBAYDI[1]

Abstract

The traditional algorithms (Prim) or (Kruskal) are able to obtain A minimum spanning tree (MST) in undirected graph easily. But many algorithms have been proposed for the purpose of obtaining spanning trees in undirected graph, these algorithms are considering the complexities of time and space. Some algorithms are generating spanning trees by using some basic cuts or circuits. In this process, the tree's cost is not considered. In this paper we will describe an algorithm for generating spanning trees in a graph of increasing cost, so we will get many possibilities, such as determining the k-th lowest spanning tree. The lowest spanning tree meets some additional constraints that may be found. We will discuss Murty's algorithm in this paper, which can find all solutions to assignment problem of increasing cost, and also discuss complexities of time and space.

Keywords: Performance Analysis, Computational Complexity, Graph, Minimum Spanning Trees.

[1] Dr., Mustansiriya University, Iraq, nadiahasan@uomustansiriyah.edu.iq

This article has been scanned by **iThenticat** No **plagiarism** detected

1. Introduction

Undirected graph (G) can be characterized as a group of (V,E), while (V) as a gathering of vertices and (E) as a gathering of edges. each two vertices are associated by edge , i.e. $E = \{(u, v) | u, v \in V\}$. The undirected weighted chart is containing a component of weighting: $w: E \to \Re$, that allocates out a weight for edges. At that point weight of edge can be likewise called as cost or distance.[1] The tree is created from sub-chart through G that has no circuits. Therefore, there is just one course from every vertex to another. in this way, the crossing tree is made out of all vertices. Also the (MST) of undirected, weighted chart (G) is the tree that having the insignificant entirety of edges' loads. There are numerous calculations that can locate a (MST) (M, for example, (Kruskal) and (Prim) calculations[2].

Kruskal's calculation: The accompanying advance must be rehashed unto the gathering M arrives at n-1 edges (initially M is unfilled). at that point we need to add the briefest edge to M, with thinking about that the edge doesn't draw up a circuit with different edges in M.

Prim's calculation: The accompanying advance must be rehashed unto the gathering M arrives at n-1 edges (initially M is emptied).[3] The most limited edge between a vertex in M and a vertex outside M ought to be Added to M (initially pick an edge with least length).

The two above calculations are diverse in the criticalness, in light of the fact that (Prim) calculation develops a tree unto it turns into the MST, while (Kruskal) calculation develops a gathering of trees unto this gathering is decreased to one tree as MST[4].

The spanning trees can be shown to by an assortment of n-1 edges. Furthermore, the edge can be shown to by unordered pair of vertices.

$$s = \{(a_1, b_1), \dots, (a_{n-1}, b_{n-1})\}$$

The character, (A) is symbolizes to all spreading over trees in the diagram G.

There are numerous calculations intended to create all spreading over trees in a diagram, for example, (Matsui, 1993; Minty, 1965; Shioura& Tamura, 1995; [5] Gabow& Myers, 1978; [6] Kapoor& Ramesh, 1995; Read &Tarjan, 1975; [7] Kapoor& Ramesh, 2000; Matsui, 1997). The complexities of good reality are the significant worries of these calculations. numerous calculations are spanning trees by utilizing some basic cuts or circuit, yet nobody considers the expense of tree during creating spanning trees.

Some calculations can produce all spanning trees without weights , for example, (Minty, 1965; Read& Tarjan, 1975),[8] so can be apply it on our concern by categoriation the trees relying upon expanding weight, in the wake of creating them. As this procedure can creates countless trees (for the most part in complete diagrams), along these lines this decision is prohibited for pragmatic purposes[9].

II Creating Spanning Trees arranged by Increasing Cost

Assuming that c(si), is the cost allocated to spanning tree, si and i is to the position of si, at that point If all spanning trees are arranged agreeing expanding cost[10].

Subsequently take on the take on that c(si) ≤ c(sj) on the off chance that i < j. The arrangement s1, s2, ... are to the arranging of spanning trees arranged by expanding cost.

By the following equation:

$$P = \{s: (i_1, j_1) \in s; \dots; (i_r, j_r) \in s; (m_1 p_1) \notin s; \dots, (m_1 p_1) \notin s\}$$

know the a portion (P) is characterized as non-void sub-bunch from all spanning trees (H) in the diagram (G). That is, it can be said that P is the gathering of spanning trees that contain all of included edges [symbolized as (i1, j1), … , (ir, jr)], and not contain any of avoided edges [symbolized as (m1, p1), … , (ml, p1)). Open edges are known as neither included nor excluded edges in segment. Will be show to the segment P as in the following equation:

$$P = \{(i_1, j_1), \dots, (i_r. j_r); (\overline{m_1, p_1}), \dots., (\overline{m_1, p_1})\}$$

The strip above shows that (m1, p1), … , (ml, pl) are prohibited edges. These edges, lead to make a few

allotments not have any spanning trees. This is known as the case (w) in the diagram G, when avoided edges are expelled or separated from the segment. So we designate the segments that are not containing any crossing trees as (vacant allotments)[11].
It ought to be referenced that (H) which shows to the gathering of all spanning trees, is likewise a segment, however the entirety of its edges are open [12]. Minimum Spanning Tree is characterized as a component of P which is made out of a spanning tree with insignificant expense and contains every included edge as it were. since each spanning tree is containing the edges (i1, j1), … , (ir, jr), in this manner the base spanning tree can be found via looking through n-r-1 from (open edges). So as to ensure that every single required edge are incorporated into a base spanning tree, they can be included before every single other edge. Likewise to ensure that nobody from rejected edges is in minimum spanning tree, they can be briefly allocated to boundless expense.

The strategy for framing partitions, ensures that the sets of included edges isn't containing any circuits. At that point Kruskal's calculation can be begun from this partial spanning tree and keep including edges.
Since the sets of included edges may not create a tree, at that point (Prim) calculation ought to be balanced by the accompanying strategy:
Include the most limited edge between a vertex inside (M) and other vertex, to M, under specification that no circuit is structure with edges of M[13]. The balanced calculation permits edges to interface two disengaged parts in spanning tree, and doesn't permit framing circuits in M.
s(P) demonstrates to the MST in partition P. What's more, c[s(P)] is its expense.
Part P by the minimum spanning tree is the topic of segmentation is the core of the algorithm proposed in this research paper[14]. When minimum spanning tree defines a section, that section can be divided into a group of resulting sections in a way that includes notifications such as the empty group is resulted through intersection of two partitions, the minimum spanning tree of main partition is not a component of any resulted partitions, in addition to the major partition is equal to the unifying of resulted partitions after subtracting its minimum spanning tree.
For more explanation, can follow the rule below: Assume that the MST in P is:

$$s(P) = \{(i_1, j_1), (i_r, j_r), (t_1, v_1), \dots, (t_{n-r-1}, v_{n-r-1})\}$$

Whereas (t1,v1),…,(tn-r-1,vn-r-1) are dissimilar from (m1,p1),…,(m1, p1). So P can be defined as the unification of single group {s(P)} and the partitions P1,…,Pn-r-1, that are not connected, where:

$$P_1 = \{(i_1, j_1), \dots, (i_r, j_r), (\overline{m_1, p_1}), \dots, (\overline{m_1, p_1}), (\overline{t_1, v_1})\}$$

$$P_1 = \{(i_1, j_1), \dots, (i_r, j_r), (t_1, v_1), (\overline{m_1, p_1}), \dots, (\overline{m_1, p_1}), .. (\overline{t_2, v_2})\}$$

$$P_1 = \{(i_1, j_1), \dots, (i_r, j_r), (t_1, v_1), (t_2, v_2), (\overline{m_1, p_1}), \dots, (\overline{m_1, p_1}), .. (\overline{t_3, v_3})\}$$

$$\dots$$

$$P_{n-r-1} = \{(i_1, j_1), \dots, (i_r, j_r), (t_1, v_1), \dots, (t_{n-r-2}, v_{n-r-2}), (\overline{m_1, p_1}), \dots, (\overline{m_1, p_1}), .., (\overline{t_{n-r-1}, v_{n-r-1}})\}$$

It can be noted that the partitions P1, …, Pn-r-1 are reciprocally disconnected by noting that any spanning tree(ST) in P either includes (t1, v1) or does not (when it is an element of P1). If it does, it either includes (t2, v2) or does not (when it is an element of P2). Taking up such this and noting that the only (ST) is containing the edges (i1, j1), …, (ir, jr), (t1, v1), …, (tn-r-1, vn-r-1) is s(P), can find that:

$$P = \{s(P)\} \cup \bigcup_{i=j}^{n-r-1} P_1$$

Each ST in partitions P1 to Pn-r-1 includes (i1,j1),…,(ir,jr) and each spanning tree does not include (m1,p1),…,(m1,p1).
Phase k in the counting process indicates to the stage when s1,...,sk are specified . In this stage, the list that contains a group of partitions M1,…,Me with the characteristics below:

1. M1, …, Me are reciprocally disconnected,
2. Every partitions in the list does not contain any spanning trees that was previously generated (su, u =

1, ..., k).

3. The unification of all partitions in the list is the set of all spanning trees that are not generated yet.

Through these characteristics, can say that[15]:

$$H = \bigcup_{u=1}^{k} \{s_i\} \cup \bigcup_{v=1}^{e} M_v$$

Also through determination a list for stage k, note that the smallest ST (k-th) is equal to s(Md), as Md is any partition in the list that

$$c[s(M_d)] = \min_{0i=1..e} \{c[s(M_i)].$$

III Design of Algorithm for organizing ST arranged by expanding cost

At the point when the graph G contains (n) vertices, the calculation proceeds in stages, until creating the littlest ST (k-th) in stage k.

In the first stage: Stage 0 is determined as equal to the partition H. then find out an minimum spanning tree for H (or for G). As in the following:

$$s_1 = \{(i_1, j_1), \dots, (i_{n-1}, j_{n-1})\}.$$

Partitions M1, …, Mn-1, that are defined as follow, will be created by (MST) of partition H.

$$M_1 = \{(i, j)\}$$
$$M_2 = \{(i, j), (i_2, j_2)\},$$
$$M_3 = \{(i, j), (i_2, j_2), (i_3, j_3)\},$$
$$\dots\dots$$
$$M_{n-1} = \{(i_1, j_1), (i_{n-2}, j_{n-2}), (\overline{i_{n-1}, j_{n-1}})\},$$

Then{M1,…,Mn-1is forming a list for stage1, so the partitions which have no spanning trees, may be taken away from the list.

In the second stage: stage (k), Since the list of stage k1 is consisted from the partitions L1,…,Lt, we count up (MST): (L1),…,s(Lt), of each partition in this list, also we count up the costs c[s(L1)],…,c[s(Lt)] for each ST.

Then, smallest ST (k-th), is the tree which has lowest cost:

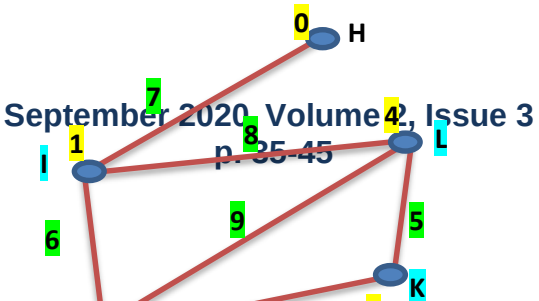
$$s_k = \left\{ s(L_i) | c[s(L_i)] = \min_{j=1..t} c\left[s(L_j)\right] \right\}.$$

(Li) represents the partition that have smallest ST of all spanning trees that are not produced yet. A list for stage k is resulted by removing (LI) from stage (k-1) list, and puting it in the partitions that are created through segmentation (Li) according s(Li). and removing blank partitions from the list. Also solving joints through choosing one partition randomly from the list and disregard others[16],[5].

.

The final stage will take the example: In this example will be explain the steps of algorithm for sorting all ST for increasing cost.

Take into consideration that the graph (G), has five vertices (H, I, J, K, L). Any ST in (G) will be composed of four edges.



September 2020, Volume 2, Issue 3
pp. 35-45

Fig. 1 Example graph (G)

The first step is to define the MST in partition (H). Then (MST) in (G) is equal to s1={(H, I), (I, J), (J, K), (K, L)} and c[s1] = 25 (see Fig. 1).

Now, G is subdivided through s1, in order to obtain four partitions, P1, …, P4, and form a list for stage 1:

$$P = \{(\overline{H, I})\}$$

$$P = \{(H, I), (\overline{I, J})\}$$

$$P = \{(H, I), (I, J), (\overline{J, K})\}$$

$$P = \{(H, I), (I, J), (J, K), (\overline{K, L})\}$$

Diagrammatically, can represent the partitions as in Figure (2) (a dotted line represents an excluded edge, and bold line represents an included edge).

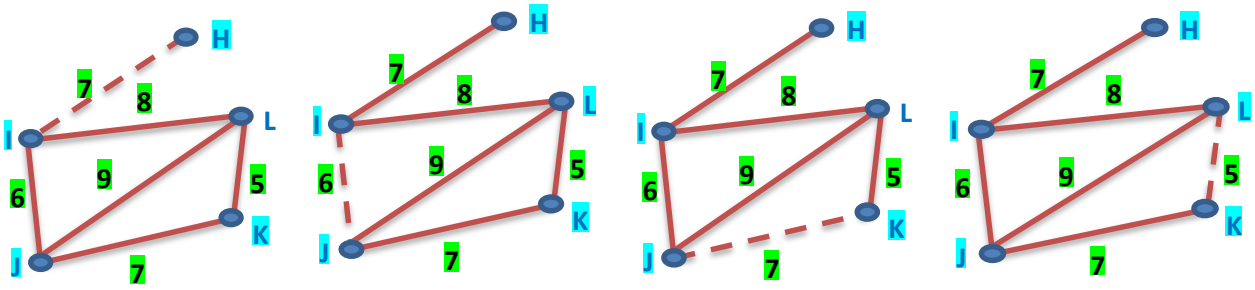


Fig. 2 Partitions P_1,…….,P_4

The next step is to count up (MST) for each partition in the list. Because P1 is disconnected, therefore it does not have (MST). The MST of P2 to P4 are:

$$s(P_2) = \{(H, I), (I, L), (L, K), (K, J)\}$$

$$s(P_3) = \{(H, I), (I, J), (I, L), (L, K)\}$$

$$s(P_4) = \{(H, I), (I, J), (J, K), (I, L)\}$$

Their costs are: $c[s(P_2)] = 27$, $c[s(P_3)] = 26$, $c[s(P_4)] = 28$

because P3 has the (MST) with lowest cost:

$$S_{2=s(P_3)=\{(H,I),(I,J),(I,L),(L,K)\}}.$$

Through subdividing (P3) according its MST s(P3) , will be gain the partitions (P31) and (P32).

$$P_{31} = \{(H, I), (I, J), (\overline{J, K}), (\overline{I, L})\}$$

$$P_{32} = \{(H, I), (I, J), (I, L), (\overline{J, K}), (\overline{L, K})\}$$

Diagrammatically, these partitions are represented in Figure3.

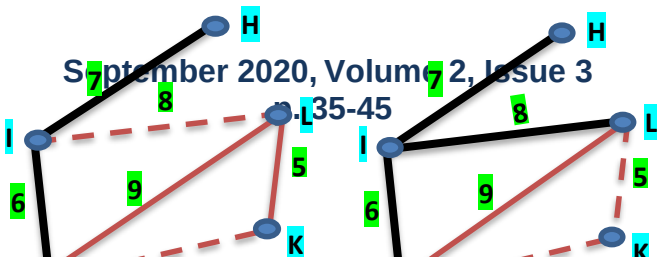


Fig. 3 Partitions P_{31} and P_{32}

The list of stage 2, is composed of {P2, P31, P4}. (P32) is removed from the list because it is disconnected. Then the MST for (P31) is:

$$s(P_{31}) = \{(H, I), (I, J), (J, L), (L, K)\}$$

With cost $c[s(P_{31})] = 27$

The list for stage 2 holds two partitions (P2 and P31) that contain a minimum spanning tree with minimum cost. links such this one can be solved by choosing any partition to be subdivided in next stage. By taking up the same steps, eight spanning trees will be gained with costs range from 25 to 31[17].

IV Implementation of the Algorithm on a computer

So as to play run out this calculation on PC, need to sort the vertexes in the list of present stage in memory. Can be shown to the partition concurring its included and barred edges. So can be represent to the particular diagram utilizing three courses of action, by represent to the head, tail and weight of the edge. successively. Furthermore, the parcel can be partition by two techniques. The first should be possible through assurance of the head and tail of included and rejected edges. The second should be possible through decide sort of each edge in the diagram, on the off chance that it is incorporated, prohibited or open. At that point the rundown of allotments can be acquired by utilizing a linked list.

The potential structure of the program that producing all ST arranged by expanding cost is:

Create ST orderly of growing cost using algorithm1:

Inputs: Graph G (V,E) & weight function (w)
Outputs: All spanning trees of Graph, arranged by expanding cost, with implementation of Output File
List = {H}
Calculation of Minimum Spanning tree(MST) of (H)
while Minimum Spanning tree(MST) not equal (∅) do
Obtain partition Ps ∈ List that includes the smallest spanning tree(SST)
Print Minimum Spanning tree(MST) of P_s to Output File
Delete P_s from List
Partition P_s.

Subdividing subroutine adds partitions to the list after inspection if they are linked and computing their (MST). The major disadvantage of this method is that either have to hold (MST) of the partition in the list (squandering memory) or recalculate it when the partition is recovered from the list (squandering time)[18]. So, the major advantage of this method is that can be keep arranged list of partitions instead of none-arranged one, and it easy to recover smallest partition. A possible program for the subdividing subroutine is:

Pseudo-Code of the PARTITION (P)
Partition($\mathbf{P_1}$) equal Partition($\mathbf{P_2}$) equal Partition($\mathbf{P_2}$)
For every edge in **P do**
if (i) not contained in **P** and not eliminate from **P then**
Make (i) eliminated from $\mathbf{P_1}$;
Make (i) contained in P_2;
Calculate Minimum Spanning tree (MST) (P_1);

V Complexities with Space and Time

|E| represents the number of edges, |V| represents the number of vertices, and N represents the number of (ST) in specific graph (G). A number of algorithms to generate all (ST) gain good time complexity through output (ST) in a specific arrangement, so can be use short notation format. And can be generate (ST) through interchange one edge from the previous (ST) in generation process[19].

By this method, can be expand the short notation format when the first SP is formed as output, and others are limited to the interchanged twin of edges.

Because no such arrangement like this is gained through this algorithm, therefore need O(N.|V|) area in order to generate all ST. Since all nodes in the list are alternately limited, the number of spanning trees imposes maximum limit on the number of partitions in the list, and the list of partitions cannot be bigger than the number of spanning trees, thus it includes a maximum of N partitions. So the partition can be delinated according the situation of its edges (open, included, or excluded). Therefore, the extent of each node is O(|E|). and the complexity's area in the partition list is O(N.|E|). In most cases, only a small part of area is needed at any moment.

The time complexity of this algorithm can be counted up depending on counting up the time complexity in algorithms of generating ST. The way to generate a spanning tree by (Kruskal) algorithm is O(|E|log |E|).

In the following statements, suppose that the partition's list is constantly preserved arranged. In this case, can be retrieve an element from the list in persistent time. Also can put an item into the list according required O(N) operations, because the maximum extent of the list is equal to the maximum number of partitions N. therefore Input and output activities will be ignored. Can implement many steps in the algorithm in steady time. such as inspection if the partition is empty or not (if connected), because this information is obtainable from the (MST) algorithm. The master loop in the algorithm can be implemented (N) times, therefore, the step (PARTITION) can be implemented (N) times. Calculate Minimum Spanning tree (MST) (P1); is O(|E|log |E|) and Add to List is O(N). The algorithm has time complexity O(N.|E|log .|E| + N2).

The complexities of time and space in our algorithm are worse than other algorithms. The algorithms which were written by mathmaticians [Gabow & Meyers (1978), Matsui (1993) and Shioura & Tamura (1995)], have ability to generate all ST of a graph in O(|V|.|E|) space and O(N.|V| |V| |E|) time. However, the aim of our algorithm is not generating all spanning trees, but to stop generating when it finds a ST that comply with some additional restrictions. This is leading to generate a small part of total number of ST.

VI Uses of Minimum Spanning Tree Problems

Possible applications can be found in the group of minimum spanning tree problems with additional restrictions, the algorithm in this paper is to generate spanning trees in order of increasing cost and inspect if

it is complying with additional restrictions. (Murty) algorithm was designed for sorting assignments according increasing cost, and it was used by (Panayiotopoulos, 1982) in comparable manner to generate an optimal solution in the issue of (Touring Salesman)[20].
In some issues, may be need to find the largest ST which complies with extra restrictions. So the algorithm can be modified in order to implement this task. For example, the two algorithms (Kruskal's) and (Prim's) can be modified to find maximum ST instead of the minimum. Also, the algorithm of generating spanning trees in order of increasing cost can be modified to generate spanning trees in order of decreasing cost[21].

VII Implementation and Experimental Results

In Computers and Software Engineering, a chart is a sort of data structure, explicitly an abstract data type (ADT) that comprises of a lot of vertexes and a lot of edges that build up connections (associations) between the nodes. This Section Presents of the implementation and the results of kruskals algorithm in C++ to find Minimum Spanning tree with Code-Blocks, in a given graph, the explanation MST as shown in Figures (4), (5) and (6).

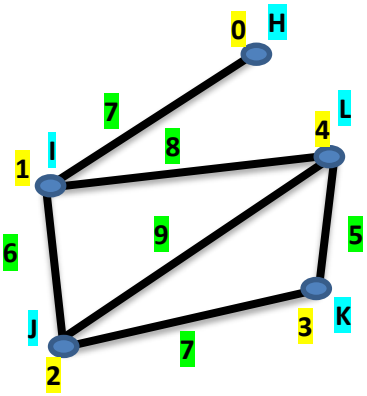


Fig.4 Connecting Weighted Graph

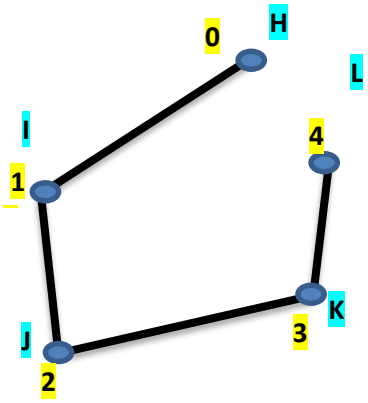
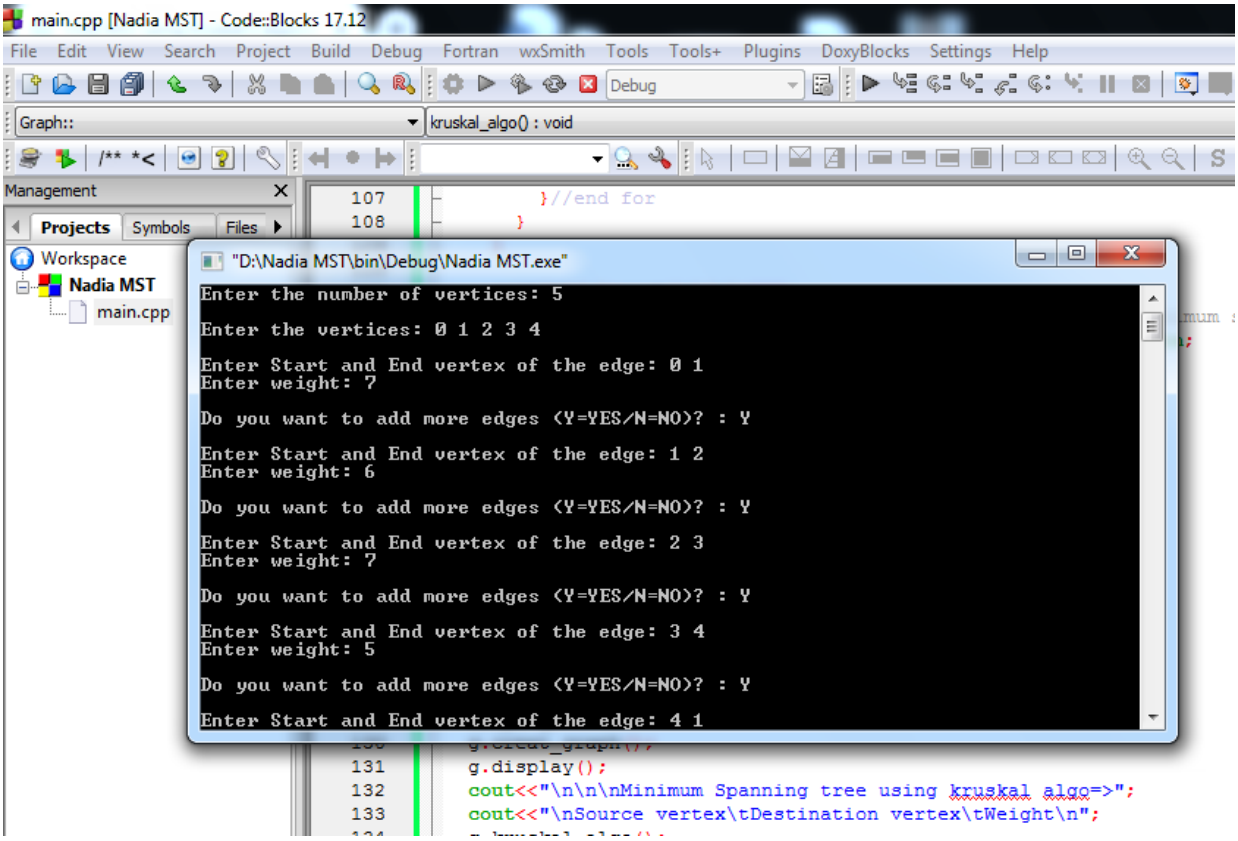


Fig.5 Minimum Spanning Tree


main.cpp [Nadia MST] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
Graph::
kruskal_algo() : void
Management
Projects Symbols Files
Workspace
Nadia MST
main.cpp
"D:\Nadia MST\bin\Debug\Nadia MST.exe"
Enter the number of vertices: 5
Enter the vertices: 0 1 2 3 4
Enter Start and End vertex of the edge: 0 1
Enter weight: 7
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 1 2
Enter weight: 6
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 2 3
Enter weight: 7
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 3 4
Enter weight: 5
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 4 1
107 }//end for
108 }
131 g.display();
132 cout<<"\n\n\nMinimum Spanning tree using kruskal algo=>";
133 cout<<"\nSource vertex\tDestination vertex\tWeight\n";

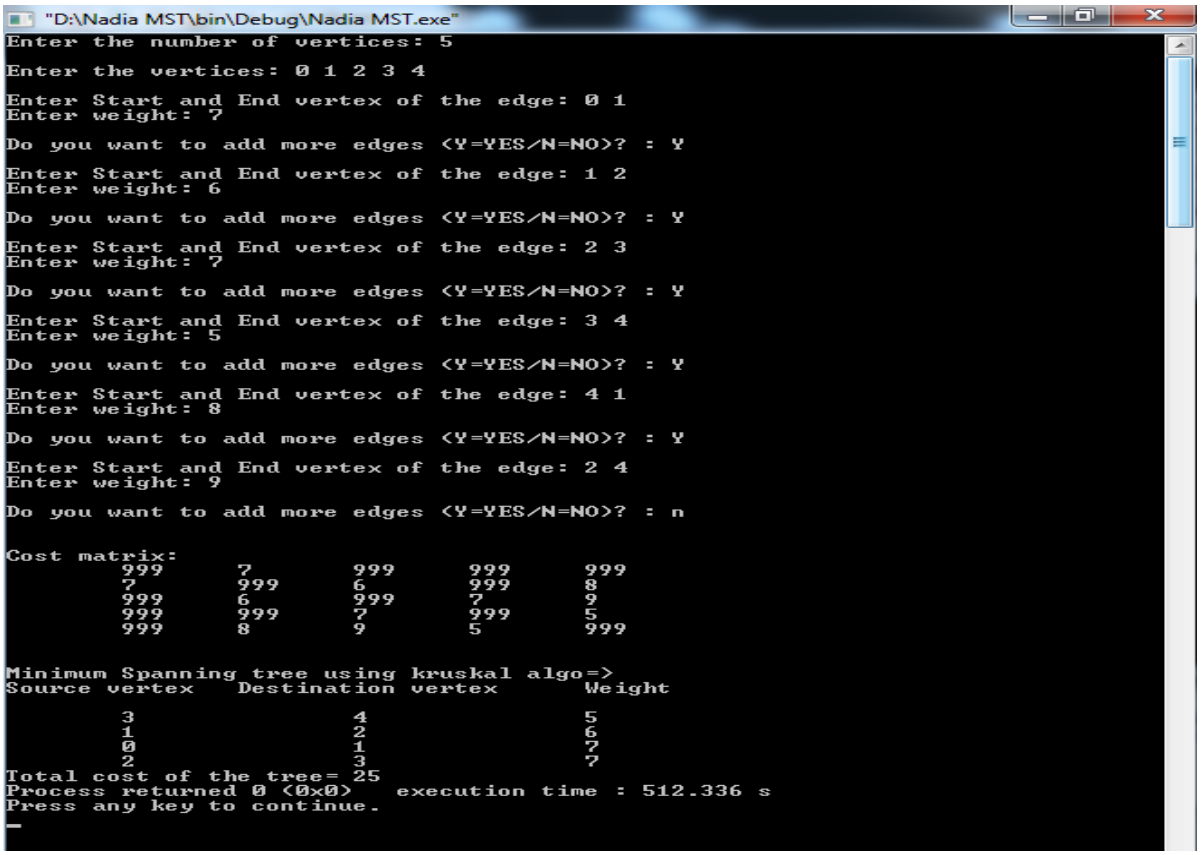

"D:\Nadia MST\bin\Debug\Nadia MST.exe"
Enter the number of vertices: 5
Enter the vertices: 0 1 2 3 4
Enter Start and End vertex of the edge: 0 1
Enter weight: 7
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 1 2
Enter weight: 6
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 2 3
Enter weight: 7
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 3 4
Enter weight: 5
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 4 1
Enter weight: 8
Do you want to add more edges (Y=YES/N=NO)? : Y
Enter Start and End vertex of the edge: 2 4
Enter weight: 9
Do you want to add more edges (Y=YES/N=NO)? : n
Cost matrix:
999 7 999 999 999
7 999 6 999 8
999 6 999 7 9
999 999 7 999 5
999 8 9 5 999
Minimum Spanning tree using kruskal algo=>
Source vertex Destination vertex Weight
3 4 5
1 2 6
0 1 7
2 3 7
Total cost of the tree= 25
Process returned 0 (0x0) execution time : 512.336 s
Press any key to continue.

Fig.7 Results of kruskals algorithm in C++ to find Minimum Spanning tree with Code-Blocks

VIII Conclusion

1. In this research paper, have been developed an algorithm to arrange all spanning trees in specific graph in order of increasing cost. so this algorithm depended on an algorithm which has been developed by the mathematician (Murty) for arranging assignments in order of increasing cost.
2. Space and time complexities are discussed briefly. And gave some instructions to execute suggested algorithm on a computer.
3. Ultimately, have been explained some possible applications of the algorithm. All these applications can be classified as minimum spanning tree problems with extra restrictions.

References

Afsana, K. & Afrida, A. & Juthi, S. (2019). A New Algorithmic Approach to Finding Minimum Spanning Tree. Conference Paper. 1-6.

Dahl, G. (1998). The 2-hop spanning tree problem. Operations Research Letters, 23. 21-26.

Sedgewick & Wayne . 2019-12-10 . Data structures and algorithms-Minimum Spanning Trees. Lecture 15a. 1-42.

Diestel, R. (1996). Graph Theory. Springer, New York, xiv + 266 pp.

Gabow, H.N. (1977). Two algorithms for generating weighted spanning trees in order. SIAM Journal on Computing, 6(1), 139-150.

Gabow, H.N. (1978). A good algorithm for smallest spanning trees with a degree constraint. Networks, 8, 201-208.

Nadia, M.H. 2015. Design And Implementation Of Shortest Path Algorithm For Network Of Roads. Journal of Engineering and Development. Vol.19, No. 06. ISSN 1813-7822. 1-12.

Gabow, H.N. & Myers, E.W. (1978). Finding all spanning trees of directed and undirected graphs. SIAM Journal on Computing, 7, 280-287.

Hall, L. & Magnanti, T. (1992). A polyhedral intersection theorem for capacitated trees. Mathematics of Operations Research, 17, 398-410.

Nadia, M. H. 2017. Analysis and Implementation of the Minimum Route Issues between some Governorates of Iraq using Bellman-Ford Algorithm. Annual Conference on New Trends in Information & Communications Technology Applications-(NTICT'2017). 1-6.

Kapoor, S. & Ramesh, H. (1995). Algorithms for enumerating all spanning trees of undirected and weighted graphs. SIAM Journal on Computing, 24, 247-265.

Kapoor, S. & Ramesh, H. (1997). An algorithm for enumerating all spanning trees of a directed graph. Algorithmica, 27(2), 120-130.

Matsui, T. (1993). An algorithm for finding all the spanning trees in undirected graphs. Technical Report METR 93-08, Dept. of Mathematical Engineering and Information Physics, University of Tokyo, Tokyo.

Matsui, T. (1997). A flexible algorithm for generating all the spanning trees in undirected graphs. Algorithmica, 18(4), 530-543.

Minty, G.J. (1965). A simple algorithm for listing all the trees of a graph. IEEE Transactions on Circuit Theory, CT-12, 120.

Murty, K.G. (1986). An algorithm for ranking all the assignments in order of increasing cost. Operations Research, 16, 682-687.

Panayiotopoulos, J-C. (1982). Probabilistic analysis of solving the assignment problem for the travelling salesman problem. European Journal of Operational Research, 9, 77-82.

Papadimitriou, C. (1978). The complexity of the capacitated tree problem. Networks, 8, 219-234.

Read, R.C. & Tarjan, R.E. (1975). Bounds on backtrack algorithms for listing cycles, paths and spanning trees. Networks, 5(3), 237-252.

Shioura, A. & Tamura, A. (1995). Efficiently scanning all spanning trees of an

undirected graph. Journal of the Operations Research Society of Japan, 38(3), 331-344.Pesquisa.

Kenneth, S.& Gerrit, K.(2005) An algorithm to generate all spanning trees of a graph in order of increasing cost